Севастопольский государственный университет
Институт информационных технологий и управления в технических системах

Сети глубокого обучения и компьютерное зрение

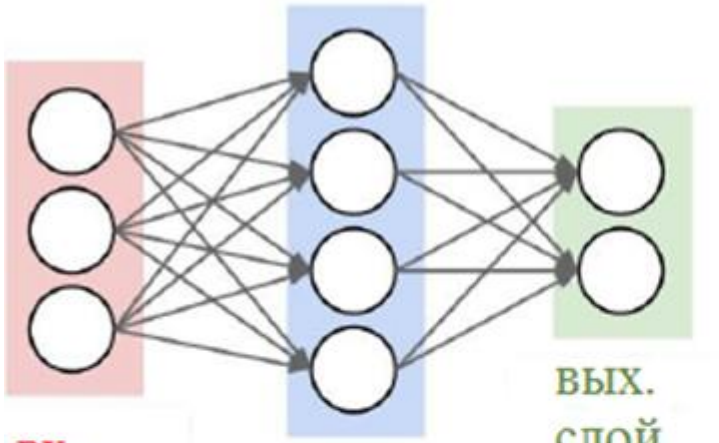
Бондарев Владимир Николаевич

Севастопольский государственный университет
Институт информационных технологий и управления в технических системах

ЛР4: Двухслойная нейросеть

Бондарев Владимир Николаевич

Двухслойная нейросеть



"2-х слойная сеть"
или "сеть с 1-м
скрытым слоем"

“Fully-connected” слои
Полносвязные слои

Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности:**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

softmax функция

Максимизировать вероятность корректного класса==минимизировать к-э

$$L_i = -\log P(Y = y_i|X = x_i), L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat	**3.2**
car	5.1
frog	-1.7

Вопрос: На этапе инициализации все s примерно равны, какие будут значения у L_i ?

Ответ: log(C), в примере log(10)≈2.3

Практика вычислений:

При вычислении Softmax функции её числитель и знаменатель могут иметь большие значения, деление больших чисел может приводить к нестабильности вычислений. Поэтому используют следующий трюк:

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+\log C}}{\sum_j e^{f_j+\log C}}$$

Мы можем выбрать любое C, обычно $\log C = -\max_j f_j$.

Т.е. мы просто сдвигаем значения элементов вектора $\boldsymbol{f}$, т.о. чтобы наибольшее возможное значение равнялось нулю.

```
f=np.array([123,456,789])  #три класса с высокими рейтингами
# сдвигаем значения эл-тов f, так чтобы наибольшее было =0
f-=np.max(f) # f=[-666,-333,0]
p=np.exp(f)/np.sum(np.exp(f)) #вычисляем softmax
```

Оптимизация

Потери – это просто функция от W.

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

$$s = f(x; W) = Wx$$

Задача найти $\nabla_W L$

Производные функции Softmax

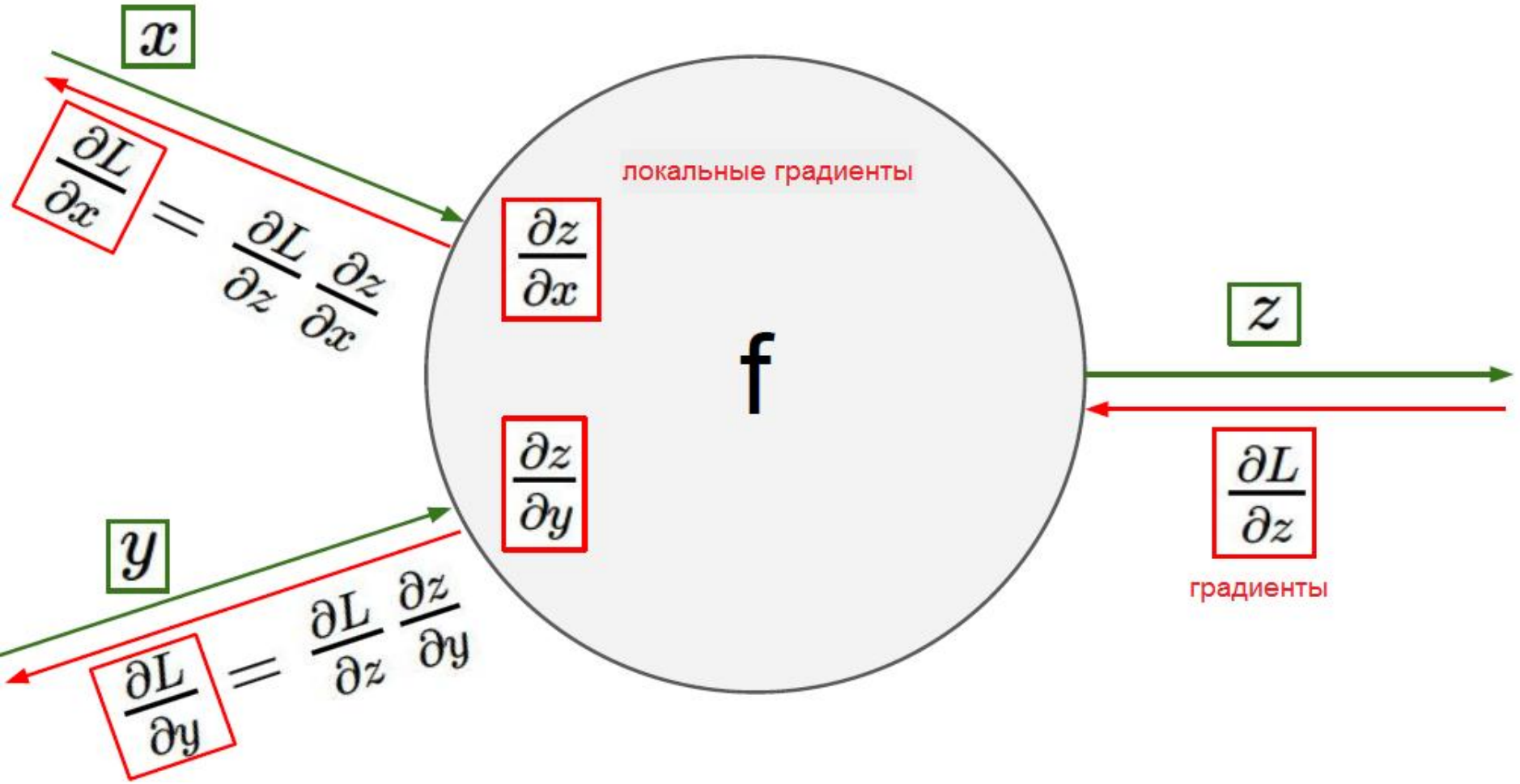
Производная функции потерь по весам корректного класса:

$$\frac{dL_i}{d\mathbf{w}_{y_i}} = (p_{y_i} - 1)\mathbf{x}_i$$

Производная функции потерь по весам любого другого класса:

$$\frac{dL_i}{d\mathbf{w}_{j \neq y_i}} = p_j \mathbf{x}_i$$

Обратное распространение (цепочное правило)

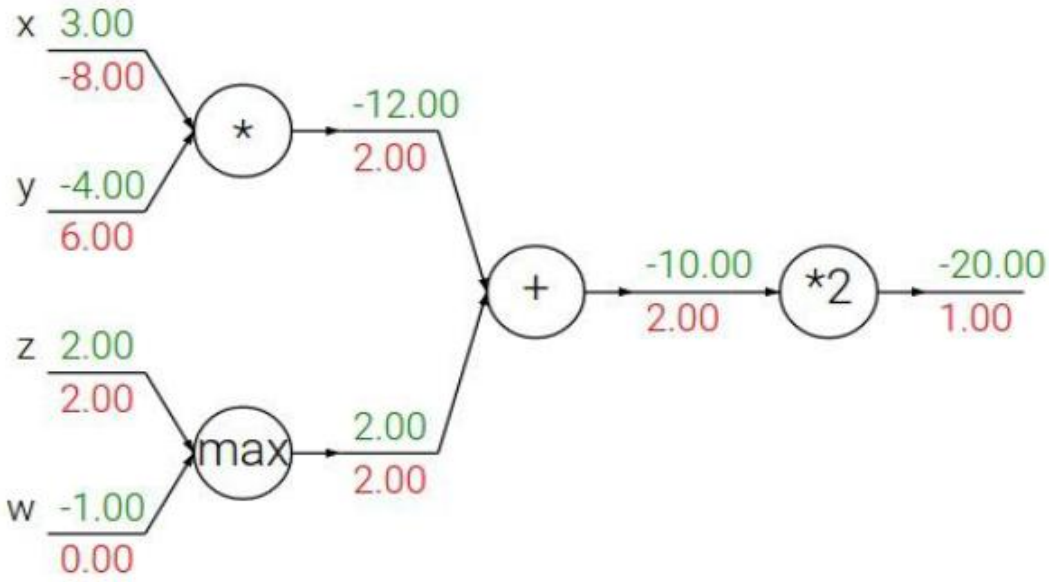


Шаблоны вычисления градиентов обратного потока

сумматор – распределитель градиента

max узел – маршрутизатор градиента

умножитель – переключатель градиента



Напоминание: Softmax (векторизованная версия)

```
# вычисление матрицы рейтингов
scores = X.dot(W) # num_train x C
# смещаем все значения в scores на -max(scores)
# для более точного вычисления функции softmax
scores -= scores.max()
# вычисляем матрицу экспонент рейтингов классов
exps = np.exp(scores)
# вероятности принадлежности классам (функция softmax)
p = exps / exps.sum(axis=1, keepdims=True)
# вычисляем средние потери по всем примерам
loss = -np.log(p[range(num_train), y]).sum() / num_train
# добавляем регуляризацию
loss += 0.5*reg * np.sum(W * W)

# вычисление градиентов функции потерь: dL_i/dw_y=(p_y-1)*x_i и dL/dw_j=(p_j*x_i
dscores = p
dscores[range(num_train), y] -= 1.0
dreg = reg * W
dW = X.T.dot(dscores) / num_train + dreg
```

Двухслойная нейросеть: Архитектура и параметры

Размерность входа сети: **NxD**
Число нейронов скрытого слоя : **H**
Число классов: **C**
Функция потерь : **-log(softmax) + L2 регуляризация**
Нелинейность скрытого слоя: **ReLU**

Архитектура сети:

вход -> полносвязный слой ->ReLU -> полносвязный слой -> softmax

Выходы второго полносвязного слоя - **рейтинги (scores)** классов.

Инициализация модели 2-х слойной сети

```
def __init__(self, input_size, hidden_size, output_size, std=1e-4):
    """
```

Входы:

- ***input_size:*** размерность входных данных - D.
- ***hidden_size:*** число нейронов скрытого слоя - H.
- ***output_size:*** число классов - C.

Веса инициализируются малыми случайными значениями, а смещения - нулями. Параметры сети сохраняются в переменной **self.params**, которая представляет собой словарь **с ключами**:

W1: Веса первого слоя; размерность (D, H)
b1: Смещения первого слоя; размерность (H,)
W2: Веса второго слоя; размерность(H, C)
b2: Смещения второго слоя; размерность (C,)

```
    """
    self.params = {}
    self.params['W1'] = std * np.random.randn(input_size, hidden_size)
    self.params['b1'] = np.zeros(hidden_size)
    self.params['W2'] = std * np.random.randn(hidden_size, output_size)
    self.params['b2'] = np.zeros(output_size)
```

Реализация прямого распространения

```
#выход слоя 1 : размер  (NxD)* (DxH)=N x H
layer1 = X.dot(W1) + b1
 #выход ReLu - (max(0,x)): размер N x H
layer1r = np.maximum(layer1, 0.0)
 #выходной слой (scores): размер (NxH)*(HxC)=N x C
scores = layer1r.dot(W2) + b2
```

Вычисление функции потерь (Softmax+L2)

```
# нормализация exp  для исключения переполнения
scores_norm = scores - scores.max()
# числитель softmax функции
exps = np.exp(scores_norm)
 # матрица вероятностей: NxC  (softmax фукнция)
p = exps / exps.sum(axis=1, keepdims=True)
#  средние потери на блоке длиной N
loss = -np.log(p[range(N), y]).sum() / N
#  полные потери (учет  L2 регуляризации)
loss += 0.5 * reg * ((W1 * W1).sum() + (W2 * W2).sum())
```

Реализация вычисления градиентов для 2-х слойной сети

```
# градиент потерь L~ -log(softmax(scores)) по scores равен p_Sy-1 или p_j,
# где p_Sy - вероятность кор. класса, p_j - вероятн. иных классов
# для вычислений dL/dscores воспользуемся матрицей вероятностей p (NxC)
dscores = p                              #  для некорректн класса = p
dscores[range(N), y] -= 1.0              #  для корректн класса = p-1
dscores /= N                             #  усредняющий множитель 1/N

#градиенты по параметрам W2, b2,W1, b1 сохраняем в словаре grads

#выходной слой: scores=np.dot(layer1r,W2)+b2 (размер N x C)
# градиент dL /db2=лок_гр * восх_гр= 1*dscores (размер C)
grads["b2"] = dscores.sum(axis=0)
# градиент dL /dW2=лок_гр * восх_гр= layer1r.T*dscores (размер HxC)
grads["W2"] = layer1r.T.dot(dscores)
#градиент dL/dlayer1r= лок_гр * восх_гр= dscores*W2.T (размер NxH)
dlayer1r = dscores.dot(W2.T)

# слой ReLU: градиент dL/dlayer1 =dL/dlayer1r, если layer1r>0 (размер NxH)
dlayer1 = dlayer1r * (layer1r > 0)       # ReLU – маршрутизатор градиента
```

Реализация вычисления градиентов для 2-х слойной сети

```
# входной слой: layer1 = X.dot(W1) + b1
# градиент dL /dW1=лок_гр * восх_гр= X.T*dlayer1 (размер DxH)
grads['W1'] = X.T.dot(dlayer1)
# градиент dL /db1=лок_гр * восх_гр= 1*dlayer1 (размер H)
grads['b1'] = dlayer1.sum(axis=0)

#добавление  градиента регуляризации
for w in ["W2", "W1"]:
        grads[w] += reg * self.params[w]
```

Формирование мини-блоков (train)

```
mask = np.random.choice(num_train,batch_size)
X_batch = X[mask]
y_batch = y[mask]
```

Обновление параметров сети (train)

```
for w in self.params:
        self.params[w] -= learning_rate*grads[w]
…
return {
    'loss_history': loss_history,
    'train_acc_history': train_acc_history,
    'val_acc_history': val_acc_history,
  }
```

Реализация предсказаний

```
layer1 = X.dot(self.params["W1"]) + self.params["b1"]                    # NxH
layer1r = np.maximum(layer1, 0.0)
scores = layer1r.dot(self.params["W2"]) + self.params["b2"]
y_pred = scores.argmax(axis=1)
```

Выбор гиперпараметров

```
results = {}
best_val = -1
learning_rates = [1e-5, …,  5e-3]
regularization_strengths = [0.05,…, 0.5]

input_size = 32 * 32 * 3
hidden_size = 80
num_classes = 10
```

Выбор гиперпараметров

```
for lr in learning_rates:
    for reg in regularization_strengths:
        net= TwoLayerNet(input_size, hidden_size, num_classes)
        stats = net.train(X_train, y_train, X_val, y_val,
            num_iters=4000, batch_size=200,
            learning_rate=lr, learning_rate_decay=0.95,
            reg=reg, verbose=True)

        val_accuracy=stats['val_acc_history'][-1]
        train_accuracy=stats['train_acc_history'][-1]

        if val_accuracy>best_val:
            best_val=val_accuracy
            best_net=net
            best_stats=stats
        results[(lr,reg)]=(train_accuracy,val_accuracy)
```

Результаты

lr 1.000000e-03 reg 2.500000e-01 train accuracy: 0.700000 val accuracy: 0.519000
lr 1.000000e-03 reg 5.000000e-01 train accuracy: 0.655000 val accuracy: 0.529000
best validation accuracy achieved during cross-validation: 0.529000

```
test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

Test accuracy: 0.534